モデル検査の実用化に関する研究

2002MT078 須田 泰寛　　2003MT052 牧晃得　　2004MT125 山本 実奈
指導教員　蜂巣 吉成

1　はじめに

近年，並行プログラムは広くで使われるようになり，設計されるプログラム自体の大規模化・複雑化も進んでいる．並行プログラムでは，同時に実行されている複数のプロセスがお互いに協調し影響を及ぼしあいながら，目的の処理をすすめていくので，実行時の挙動を実行時に把握することは難しく，逐次プロセスでは起こることのなかった，デッドロックやスタベーションといった不具合が発生する可能性がある．

また，これらの不具合は設計時での発見は困難で，モデル検査をおこなう事で人為的な誤りを発見し，そこから発生する故障 (fault) をなくすことができれは開発において大きな利点となる．実装後に障害 (failure) が発見された場合の手戻りの発生を防ぐことができるからである．

従来では，故障を引き起こす欠陥の発見にテストを用いる方法がある．ただし，並行プログラムの場合は同一の入力に対して常に同じ結果が得られるわけではないので，テストによる故障の発見が困難な場合がある．

そこで，ソフトウェアの設計時にモデル検査をおこない，早期に設計の間違いを発見できれば開発期間を短縮し開発コストの削減にもつながる．

設計は UML を用いて表される事が多く，一般にそのままではモデル検査を行えない．また，UML 記述をモデル検査のためにモデル検査用言語のコードに変換するのは手間が大きい．また，モデル検査用の言語はあまり知られておらず習得が困難である．

それらがモデル検査の実用化の障害となっている．さらに，モデル検査によって検出されたエラーから元の設計の故障箇所を特定するのにもモデル検査用言語の知識が必要であり，実際にモデル検査をおこなうことは困難である．

本研究ではモデル検査言語に詳しくない技術者にも容易に実行前検査がおこなえるような支援技術の開発を目的とする．

具体的には，UML のステートマシン図，シーケンス図からモデル検査用の入力コードを生成するツールと．検査結果から間違いのある箇所を UML 図に表示するツールの設計をおこなう．

研究の手順を以下に示す．

- 使用するモデル検査ツールの比較検討
- 検査対象と自動生成する部分の検討
- UML 図と検査ツール用の言語との対応づけ
- モデル検査結果と UML 図の対応づけ

2　モデル検査ツールの比較

2.1　モデル検査の概要

モデル検査とは，検査対象の有限状態遷移機械の状態の直積を取ってモデル化し，システムの状態空間を作り，検査対象の状態を自動的かつ網羅的に探索することで安全性を検査する技術である．

以下，モデル検査ツールとして広く利用されている，ペトリネット，SPIN，FDR について比較する．

2.2　ペトリネット

ペトリネット [5] では複数の状態遷移機械を 1 つの図で記述可能であり，記述した図をシミュレーションにより実行することができる．プレースと呼ばれる円，トランジションと呼ばれる棒，トランジションとプレースを結ぶ有向アーク，トークンと呼ばれる点で構成され，トランジションで同期を表現する．

2.3　SPIN

SPIN[1] とは Bell 研究所の G.J.Holzamann によって開発されたモデル検査ツールであり，Simple Promela Interpreter の略称である．

SPIN では Promela(Process Meta Language) という検査ツール言語を使い，並列動作をモデル化することができる．Promela はチャネル通信オートマトンを記述するための言語で，C 言語から多くの構文を取り入れている．

2.4　FDR

FDR[2] とは Failures Divergence Refinement の略称である．FDR は CSP(Communicating Sequential Processe) 理論に基づく並行システム検証ツールである．仕様の検証をはじめ，デッドロックやスタベーションの検出をすることもできる．CSP は，C.A.R Hoare が考案したモデル検査言語で，プロセス代数に基づき，ソフトウェアをモデル化する．

2.5　ツールの比較

ペトリネットによる記述には，一般的に図が大きくなり理解性が低くなるという欠点があるので，本件研究では用いない．

SPIN と FDR は共に状態空間を作り，圧縮後検査をしている．

ステートマシン図から記述する場合は、CSP 記述より Promela 記述の方が変換しやすく，検査結果がわかりやすい．

本研究ではステートマシン図により近い SPIN を使用する．

3　ソフトウェアの実行前検査

3.1　工程

モデル検査の工程を**図 1** に表す。本研究の目的は四角の部分である。モデル検査言語自動生成ツールでは，モデル検査言語を自動生成する．デバックツールでは，モデル検査の結果がエラーの場合エラーの場所を UML 図に表す．

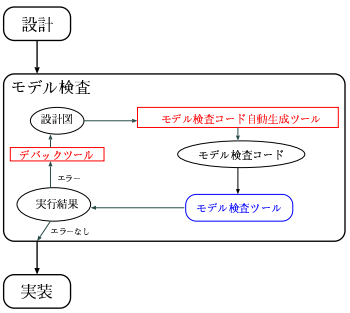

図 1　モデル検査の工程

3.2　検査項目

本研究では重要な誤りとして，デッドロック，スタベーションの発生の有無を検査し，発生した場合には発生箇所の特定をおこなう．

3.3　入力対象

ステートマシン図は状態遷移機械の状態遷移と振る舞いを表現している．ステートマシン図の構成要素として，状態，遷移がある．遷移を表現する矢印には，イベント，ガード条件，アクションを生成することができる．

シーケンス図は動的な振舞いを記述するのに適している．オブジェクト間の通信を記述することができ，どのようなメッセージをうけて振る舞いを行うのかを明確に表す事ができるので，アクションの実行順序を表現するのに適している．

ステートマシン図だけでは，アクションの送信元と送信先の情報が不十分であるので，それらの情報をシーケンス図でおぎなう必要がある．したがってツールの入力対象をステートマシン図とシーケンス図とした．それぞれを**図 2** に示す．

3.4　入力対象と Promela の対応

Promela 記述

Promela は主に**図 3** に示す 4 つの部分から構成される．mtype 部では全てのイベント型を宣言し，proctype 部では状態遷移機械の振る舞いを記述する．user 部では初期イベントを送信し，init 部ではインスタンスの宣言をする．シーケンス図ではアクションの記述をおこない，初期状態は一番初めに記述された状態とする．また，状態遷移機械同士のイベントの送信はチャネルを使っておこなう．

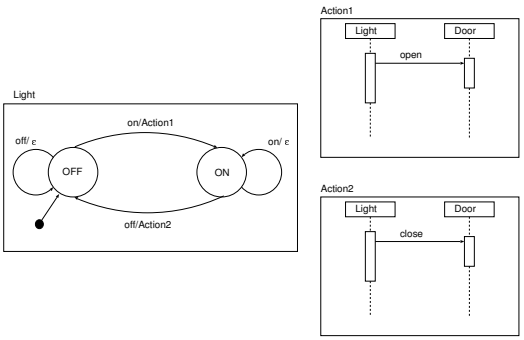

図 2　ステートマシン図，シーケンス図

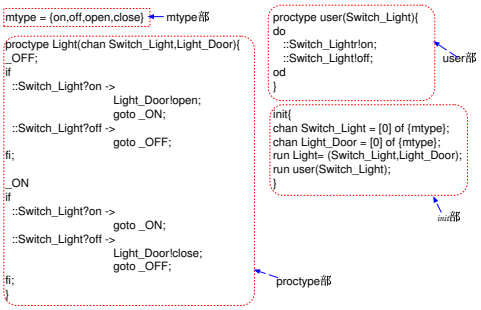

図 3　Promela 記述

Promela の生成

メッセージ送信は ‘送信チャネル名!イベント名’ のように記述し，メッセージ受信は ‘受信チャネル名?イベント名’ のように記述する．また，チャネルは状態遷移機械ごとに作成し，‘イベント送信元 STM 名_イベント受信先 STM 名’ のように記述する．それらの対応を**図 4** に示す．

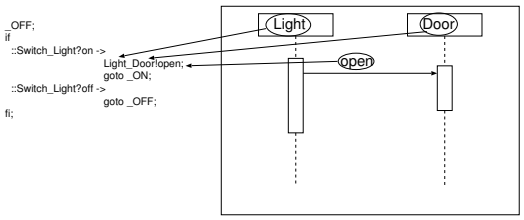

図 4　チャネル名

Promela のテンプレートに UML ツールのステートマシン図とシーケンス図からイベント名，チャネル名，状態遷移機械名を当てはめることで検査用の Promela コードを生成できる．

Promela のテンプレートを**図 5** に示す．

3.5　エラーのデバック

エラーのデバック方法

Promela を SPIN で実行すると，モデル検査用 C プログラムを自動生成する．検査用 C プログラムをコンパイルし，実行すると，モデル検査の結果を表示する．この

```
mtype ={イベント名1,イベント名2・・・}

proctype 状態遷移機械名= (受チャネル名1,送チャネル名1,・・・){
初期状態名前:
if
  ::受チャネル?受イベント名 ->
  ::送チャネル!送イベント名 ;
goto 遷移先状態名;
  ::受チャネル?受イベント名 ->
goto 遷移先状態名;
fi;
}

状態名
if
  ::受チャネル?受イベント名 ->
  ::送チャネル!送イベント名 ;
goto 遷移先状態名;
  ::受チャネル?受イベント名 ->
goto 遷移先状態名;
fi;
}

・・・
```

```
proctype user(送チャネル名1,送チャネル名2,・・・)
do
  ::送チャネル名!送イベント名
  ・・・
od

init{
chan チャネル名=[0] {mtype};
・・・
run 状態遷移機械名(送チャネル名,受チャネル名)
・・・
}
```

図 5　Promela テンプレート

時, エラーが発見されると反例が生成される. 反例にはエラーのおこる実行列が表示される.

モデル検査ツールではデッドロックやスターベーションのおきた箇所を指摘するが，エラーのおきた箇所以外にも，誤りの原因が存在する場合がある．そのような場合には，モデル検査ツールで自動的に間違いの原因の箇所を特定することはできない．実際にエラーの原因を特定するには，一般に反例などをみて，検査する人間が判断する．反例など SPIN の実行結果を読み取るさいには Promela の知識が必要となるが，本研究では Promela に詳しくない技術者にもエラー箇所を特定できるよう，UML 図にエラーを表示する方法を提案する．

入力対象

Spin ではいろいろなシミュレーションをすることが出来るが，ランダムに実行したものは，実行毎に結果が異なり，必ずしもエラーのおこる実行列を表示するとは限らない．一方，反例には必ずエラーのおこる実行列が表示される．本研究では，反例を入力対象とし，エラー表示方式の検討をおこなう．

表示対象

Spin で表示されたエラー箇所を UML 図にあらわしたい．反例には，エラーのおきる実行列 (行数，プロセス名，状態) が含まれる．ただし，状態は自動的に数字がふられていて，具体的にどの状態か特定することが出来ない．ここで，promela の実行文から，UML のステートマシン図，シーケンス図の箇所を特定することを考える．ステートマシン図は，シーケンス図の箇所を特定するのに必要な構成要素をあげる．

- ステートマシン図の構成要素
  - 状態，遷移 (イベントトリガ，アクション)
- シーケンス図の構成要素
  - ステートマシン名，メッセージ

状態が特定できないので，ステートマシン図上の箇所の特定は出来ないと考えた．

promela 記述のメッセージの送受信から，シーケンス図上の箇所を特定することができると考えた．また，シーケンス図は流れを表しているので，エラーのおこる軌跡を表現しやすいと考えた．以上より，エラーをシーケンス図にあらわすことにした．

表示方法

シーケンス図の間違いのおこる軌跡に含まれるメッセージ部分に色づけすることで，エラーを表示することを考えた．UML ツール EA(Enterprise Architect)[4] の XMI データの入出力機能を使う．UML 図を XMI データとして出力し，XMI データを書き換え，EA に再び読み込ませることでエラーを表示できると考えた.

promela 記述とシーケンス図の対応

例として，デッドロックをおこすプリンタとスキャナの例をあげる．ステートマシン図を**図 6**，シーケンス図を**図 7** に示す.

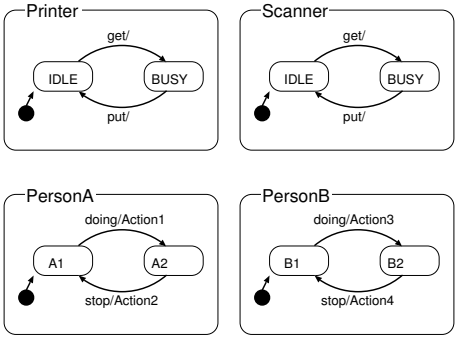

図 6　プリンタとスキャナの例のステートマシン図

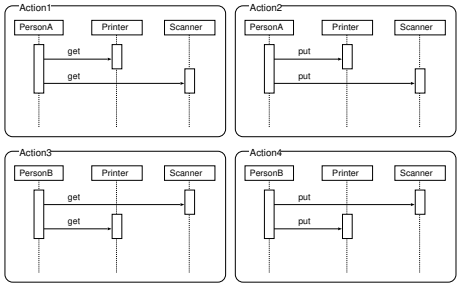

図 7　プリンタとスキャナの例のシーケンス図

反例は**図 8** のように表示される.

図 8 から, ‘personB_scanner!get’ というメッセージ通信を例にあげる．本研究では，チャネル名は，‘送信プロセス名_受信プロセス名’ としている．ステートマシンごとにプロセスを生成するので，プロセス名はステートマシン名と考える. ここから, 送信先 STM 名と受信先 STM 名が特定でき，‘personB_scanner!get’ は，‘personB’ から ‘scanner’ に ‘get’ というメッセージを送ったという意味だとわかる．‘personB_?printer’ も同様である．**図 9** の網掛けの部分のメッセージであると特定できる.

複数のアクションで全く同じメッセージを送信する場合を考える. 例えば, **図 10** のようなときである.

‘personA’ から ‘printer’ へ ‘get’ を送るメッセージが複数ある．**図 10** の網掛けの部分である．この場合，‘personA_printer!get’ という promela 記述から，どちら

```
Starting :init: with pid 0
spin: warning, "PS2.prom" is newer than PS2.prom.trail
Starting personA with pid 1
  1:   proc  0 (:init:) line  81 "PS2.prom" (state 1)  [(run personA(personA_event,personA_printer,personA_scanner))]
Starting personB with pid 2
  2:   proc  0 (:init:) line  82 "PS2.prom" (state 2)  [(run personB(personB_event,personB_printer,personB_scanner))]
Starting printer with pid 3
  3:   proc  0 (:init:) line  83 "PS2.prom" (state 3)  [(run printer(personA_printer,personB_printer))]
Starting scanner with pid 4
  4:   proc  0 (:init:) line  84 "PS2.prom" (state 4)  [(run scanner(personA_scanner,personB_scanner))]
Starting user with pid 5
  5:   proc  0 (:init:) line  85 "PS2.prom" (state 5)  [(run user(personA_event,personB_event))]
  6:   proc  5 (user) line  67 "PS2.prom" (state 1)   [personA_event!start]
  7:   proc  1 (personA) line  36 "PS2.prom" (state 1) [personA_event?start]
  8:   proc  5 (user) line  68 "PS2.prom" (state 2)   [personB_event!start]
  9:   proc  2 (personB) line  52 "PS2.prom" (state 1) [personB_event?start]
 10:   proc  2 (personB) line  53 "PS2.prom" (state 2) [personB_scanner!get]
 11:   proc  3 (printer) line   7 "PS2.prom" (state 3) [personB_printer?get]
 12:   proc  2 (personB) line  54 "PS2.prom" (state 3) [personB_printer!get]
 13:   proc  4 (scanner) line  22 "PS2.prom" (state 3) [personB_scanner?get]
spin: trail ends after 13 steps
#processes: 6
 13:   proc  5 (user) line  67 "PS2.prom" (state 3)
 13:   proc  4 (scanner) line  27 "PS2.prom" (state 11)
 13:   proc  3 (printer) line  11 "PS2.prom" (state 11)
 13:   proc  2 (personB) line  59 "PS2.prom" (state 11)
 13:   proc  1 (personA) line  37 "PS2.prom" (state 2)
 13:   proc  0 (:init:) line  86 "PS2.prom" (state 6) <valid end state>
6 processes created
```

図 8　プリンタとスキャナの例の反例

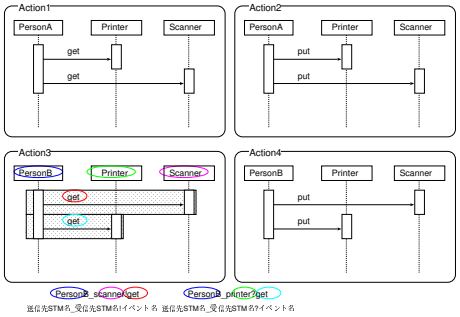

図 9　シーケンス図と promela 記述の対応

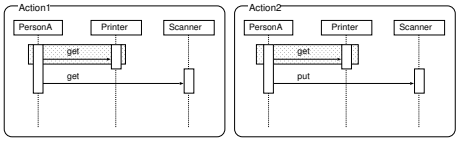

図 10　同じメッセージを複数送信する例

のアクションのメッセージか特定できない．

反例には promela の実行文の行数が書かれているので，promela 記述上の箇所は特定できる．どのアクションのイベントかわかれば，メッセージが一意に特定できる．状態遷移機械を promela で生成するさいに，アクションの情報を付加することで解決できる．

4　考察

本研究で検査する項目はデッドロックとスタベーションである．提案したツールによってデッドロックの検査をおこなうことができる．しかし，スタベーションの検査は現状では出来ない．

4.1　ラベルについて

スタベーションの検査をするためには，ラベルをつける必要がある．progress ラベルを通らない軌跡がある場合エラーと判断される．また，end ラベルで終了する場合は正常終了とみなされる．ラベルをつける箇所を細かく指定しなければ正常な検査が出来ないので，シーケンス図に表すようにする．

4.2　エラー表示の有用性についての考察

反例について

反例に表示される実行列にメッセージの送受信が一つもない場合がある．その場合，シーケンス図に表示するエラーの軌跡がないので，エラーの箇所を探すのに判断材料がない．人間が検査する場合，他のシミュレーションをしてエラー箇所を探すので，UML 図に表示される情報では不十分である．チャネルを使用してメッセージを送受信するように Promela 記述を生成してエラーの軌跡を表示する．

また，逆にメッセージの送受信がたくさんあった場合，シーケンス図のほとんどのメッセージがエラー表示されてしまう．実行の順番を推測することが困難となり，エラー箇所の特定が難しくなる．メッセージの送受信が多いのでシーケンス図ごとに細分化してエラー箇所が特定できると考える．

シーケンス図について

本研究では，アクション一つ一つに個別のシーケンス図を作っている．そのために，量が多くなると全体の流れが分かりにくい．また，個別のシーケンス図に戻すので，シーケンス図を見ても実行した順番がわかりにくい．ステートマシン図ごとにシーケンス図をまとめてエラー箇所の特定ができると考える．

5　まとめ

本研究では，UML を理解していれば実行前検査が出来るような方法を提案し，実行前検査支援ツールの設計をおこなった．UML のステートマシン図, シーケンス図と，promela との対応づけをおこない，コードを自動生成する方法を提案した．また，Spin によるモデル検査の実行結果から得られる情報と, ステートマシン図，シーケンス図との対応づけをおこない，エラーを UML 図に表示する方法を提案した．

今後の課題として，ラベルの問題の解決，またツールの作成があげられる．

参考文献

[1] Alcatel-Lucent：Basic Spin Manual, http://spinroot.com/spin/Man/Manual.html/

[2] Formal Systems (Europe) Ltd,：FDR2 User Manual, http://www.fsel.com/documentation/fdr2/html/

[3] C.A.R. Hoare：Communicating Seqential Processes, Prentice-Hall Internationl Ltd., 1985

[4] Sparx Systems Japan Co., Ltd,：UML モデリングツール Enterprise Architect, http://www.sparxsystems.jp/

[5] 青山幹雄，内平直志，平石邦彦：ペトリネットの理論と実践，朝倉書店，1995